Topics in Logic and Complexity
Lecture 1

Anuj Dawar

MPhil Advanced Computer Science, Lent 2013

What is This Course About?

Complexity Theory is the study of what makes some algorithmic problems inherently difficult to solve.

Difficult in the sense that there is no *efficient* algorithm.

Mathematical Logic is the study of formal mathematical reasoning.

It gives a *mathematical* account of meta-mathematical notions such as *structure*, *language* and *proof*.

In this course we will see how logic can be used to study complexity theory. In particular, we will look at how complexity relates to *definability*.

Computational Complexity

Complexity is usually defined in terms of *running time* or *space* asymptotically required by an algorithm. *E.g.*

- Merge Sort runs in time $O(n \log n)$.
- Any sorting algorithm that can sort an arbitrary list of n numbers requires time $\Omega(n \log n)$.

Complexity theory is concerned with the hardness of *problems* rather than specific algorithms.

We will mostly be concerned with *broad* classification of complexity: *logarithmic* vs. *polynomial* vs. *exponential*.

Graph Properties

For simplicity, we often focus on *decision problems*.

As an example, consider the following three decision problems on *graphs*.

1. Given a graph $G = (V, E)$ does it contain a *triangle*?
2. Given a directed graph $G = (V, E)$ and two of its vertices $s, t \in V$, does G contain a *path* from s to t?
3. Given a graph $G = (V, E)$ is it *3-colourable*? That is,

   is there a function $\chi : V \to \{1, 2, 3\}$ so that whenever $(u, v) \in E$, $\chi(u) \neq \chi(v)$.

Graph Properties

1. Checking if G contains a triangle can be solved in *polynomial time* and *logarithmic space*.

2. Checking if G contains a path from s to t can be done in *polynomial time*.

Can it be done in *logarithmic space*?

Unlikely. It is NL-complete.

3. Checking if G is 3-colourable can be done in *exponential time* and *polynomial space*.

Can it be done in *polynomial time*?

Unlikely. It is NP-complete.

Logical Definability

In what kind of formal language can these decision problems be *specified* or *defined*?

The graph $G = (V, E)$ contains a triangle.

$$\exists x \in V\, \exists y \in V\, \exists z \in V (x \neq y \wedge y \neq z \wedge x \neq z \wedge E(x,y) \wedge E(x,z) \wedge E(y,z))$$

The other two properties are *provably* not definable with only first-order quantification over vertices.

Second-Order Quantifiers

3-Colourability and *reachability* can be defined with quantification over *sets of vertices*.

$$\begin{aligned}
&\exists R \subseteq V\, \exists B \subseteq V\, \exists G \subseteq V \\
&\qquad \forall x (Rx \vee Bx \vee Gx) \wedge \\
&\qquad \forall x (\neg(Rx \wedge Bx) \wedge \neg(Bx \wedge Gx) \wedge \neg(Rx \wedge Gx)) \wedge \\
&\qquad \forall x \forall y (Exy \to (\neg(Rx \wedge Ry) \wedge \\
&\qquad\qquad \neg(Bx \wedge By) \wedge \\
&\qquad\qquad \neg(Gx \wedge Gy)))
\end{aligned}$$

$$\forall S \subseteq V (s \in S \wedge \forall x \forall y ((x \in S \wedge E(x,y)) \to y \in S) \to t \in S)$$

Course Outline

This course is concerned with the questions of (1) how definability relates to computational complexity and (2) how to analyse definability.

1. Complexity Theory—a review of the major complexity classes and their interrelationships (3L).
2. First-order and second-order logic—their expressive power and computational complexity (3L).
3. Lower bounds on expressive power—the use of games and locality (3L).
4. Fixed-point logics and descriptive complexity (3L).
5. Monadic Second-Order Logic and Automata (4L).

Useful Information

Some useful books:

- C.H. Papadimitriou. Computational Complexity. Addison-Wesley. 1994.
- S. Arora and B. Barak. Computational Complexity. CUP. 2009.
- H.-D. Ebbinghaus and J. Flum. Finite Model Theory (2nd ed.) 1999.
- N. Immerman. Descriptive Complexity. Springer. 1999.
- L. Libkin. Elements of Finite Model Theory. Springer. 2004.
- E. Grädel et al. Finite Model Theory and its Applications. Springer. 2007.

Course website: `http://www.cl.cam.ac.uk/teaching/1213/L15/`

Decision Problems and Algorithms

Formally, a *decision problem* is a set of strings $L \subseteq \Sigma^*$ over a finite alphabet Σ.

The problem is *decidable* if there is an *algorithm* which given any input $x \in \Sigma^*$ will determine whether $x \in L$ or not.

The notion of an *algorithm* is formally defined by a *machine model*: A *Turing Machine*; *Random Access Machine* or even a *Java program*.

The choice of machine model doesn't affect what is or is not decidable.

Similarly, we say a function $f : \Sigma^* \to \Delta^*$ is *computable* if there is an algorithm which takes input $x \in \Sigma^*$ and gives output $f(x)$.

Turing Machines

For our purposes, a Turing Machine consists of:

- K — a finite set of states;
- Σ — a finite set of symbols, including $\sqcup$.
- $s \in K$ — an initial state;
- $\delta : (K \times \Sigma) \to (K \cup \{a, r\}) \times \Sigma \times \{L, R, S\}$

  A transition function that specifies, for each state and symbol a next state (or accept a or reject r), a symbol to overwrite the current symbol, and a direction for the tape head to move (L – left, R – right, or S - stationary)

Configurations

A complete description of the configuration of a machine can be given if we know what state it is in, what are the contents of its tape, and what is the position of its head. This can be summed up in a simple triple:

Definition

A *configuration* is a triple (q, w, u), where $q \in K$ and $w, u \in \Sigma^\star$

The intuition is that (q, w, u) represents a machine in state q with the string wu on its tape, and the head pointing at the last symbol in w.

The configuration of a machine completely determines the future behaviour of the machine.

Computations

Given a machine $M = (K, \Sigma, s, \delta)$ we say that a configuration (q, w, u) *yields in one step* (q', w', u'), written

$$(q, w, u) \to_M (q', w', u')$$

if

- $w = va$;
- $\delta(q, a) = (q', b, D)$; and
- either $D = L$ and $w' = v$ $u' = bu$
  or $D = S$ and $w' = vb$ and $u' = u$
  or $D = R$ and $w' = vbc$ and $u' = x$, where $u = cx$. If u is empty, then $w' = vb\sqcup$ and u' is empty.

Computations

The relation $\to_M^\star$ is the reflexive and transitive closure of $\to_M$.

A sequence of configurations $c_1, \ldots, c_n$, where for each i, $c_i \to_M c_{i+1}$, is called a *computation* of M.

The language $L(M) \subseteq \Sigma^\star$ *accepted* by the machine M is the set of strings

$$\{x \mid (s, \triangleright, x) \to_M^\star (\mathrm{acc}, w, u) \text{ for some } w \text{ and } u\}$$

A machine M is said to *halt on input* x if for some w and u, either $(s, \triangleright, x) \to_M^\star (\mathrm{acc}, w, u)$ or $(s, \triangleright, x) \to_M^\star (\mathrm{rej}, w, u)$

Complexity

For any function $f : \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{TIME}(f(n))$ if there is a machine $M = (K, \Sigma, s, \delta)$, such that:

- $L = L(M)$; and
- The running time of M is $O(f(n))$.

Similarly, we define $\mathsf{SPACE}(f(n))$ to be the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n.

In defining space complexity, we assume a machine M, which has a read-only input tape, and a separate work tape. We only count cells on the work tape towards the complexity.

Nondeterminism

If, in the definition of a Turing machine, we relax the condition on δ being a function and instead allow an arbitrary relation, we obtain a *nondeterministic Turing machine*.

$$\delta \subseteq (K \times \Sigma) \times (K \cup \{a, r\} \times \Sigma \times \{R, L, S\}).$$

The yields relation $\to_M$ is also no longer functional.

We still define the language accepted by M by:

$$L(M) = \{x \mid (s, \triangleright, x) \to_M^\star (\mathrm{acc}, w, u) \text{ for some } w \text{ and } u\}$$

though, for some x, there may be computations leading to accepting as well as rejecting states.

Nondeterministic Complexity

For any function $f : \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathsf{NTIME}(f(n))$ if there is a *nondeterministic* machine $M = (K, \Sigma, s, \delta)$, such that:

- $L = L(M)$; and
- The running time of M is $O(f(n))$.

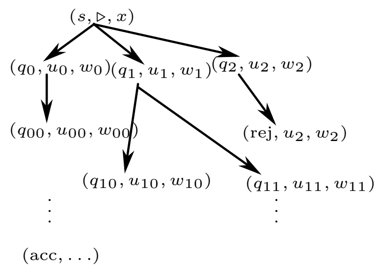
The last statement means that for each $x \in L(M)$, there is a computation of M that accepts x and whose length is bounded by $O(f(|x|))$.

Similarly, we define $\mathsf{NSPACE}(f(n))$ to be the languages accepted by a *nondeterminstic* machine which uses $O(f(n))$ tape cells on inputs of length n.

As before, in reckoning space complexity, we only count work space.

Computation Trees

With a nondeterministic machine, each configuration gives rise to a tree of successive configurations.



Complexity Classes

A complexity class is a collection of languages determined by three things:

- A model of computation (such as a deterministic Turing machine, or a nondeterministic TM, or a parallel Random Access Machine).
- A resource (such as time, space or number of processors).
- A set of bounds. This is a set of functions that are used to bound the amount of resource we can use.

Polynomial Bounds

By making the bounds broad enough, we can make our definitions fairly independent of the model of computation.

> The collection of languages recognised in *polynomial time* is the same whether we consider Turing machines, register machines, or any other deterministic model of computation.
>
> The collection of languages recognised in *linear time*, on the other hand, is different on a one-tape and a two-tape Turing machine.

We can say that being recognisable in polynomial time is a property of the language, while being recognisable in linear time is sensitive to the model of computation.

Reading List for this Part

1. Papadimitriou. Chapters 1 and 2.
2. Arora and Barak. Chapter 1.
3. Grädel et al. Chapter 1 (Weinstein).

Topics in Logic and Complexity Part 2

Anuj Dawar

MPhil Advanced Computer Science, Lent 2013

Polynomial Time Computation

$$\mathsf{P} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(n^k)$$

The class of languages decidable in polynomial time.

The complexity class P plays an important role in complexity theory.

- It is robust, as explained.
- It serves as our formal definition of what is *feasibly computable*

Nondeterministic Polynomial Time

$$\mathsf{NP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(n^k)$$

That is, NP is the class of languages accepted by a *nondeterministic* machine running in polynomial time.

Since a deterministic machine is just a nondeterministic machine in which the transition relation is *functional*, $\mathsf{P} \subseteq \mathsf{NP}$.

Succinct Certificates

The complexity class NP can be characterised as the collection of languages of the form:

$$L = \{x \mid \exists y\, R(x,y)\}$$

Where R is a relation on strings satisfying two key conditions

1. R is decidable in polynomial time.
2. R is *polynomially balanced*. That is, there is a polynomial p such that if $R(x,y)$ and the length of x is n, then the length of y is no more than $p(n)$.

Equivalence of Definitions

If $L = \{x \mid \exists y\, R(x,y)\}$ we can define a nondeterministic machine M that accepts L.

The machine first uses nondeterministic branching to *guess* a value for y, and then checks whether $R(x,y)$ holds.

In the other direction, suppose we are given a nondeterministic machine M which runs in time $p(n)$.

Suppose that for each $(q,\sigma) \in K \times \Sigma$ (i.e. each state, symbol pair) there are at most k elements in $\delta(q,\sigma)$.

Equivalence of Definitions

For y a string over the alphabet $\{1,\ldots,k\}$, we define the relation $R(x,y)$ by:

- $|y| \leq p(|x|)$; and
- the computation of M on input x which, at step i takes the "$y[i]$th transition" is an accepting computation.

Then, $L(M) = \{x \mid \exists y\, R(x,y)\}$

Space Complexity Classes

$\mathsf{L} = \mathsf{SPACE}(\log n)$

The class of languages decidable in logarithmic space.

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

The class of languages decidable by a nondeterministic machine in logarithmic space.

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$

Inclusions between Classes

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

Of these, the following are direct from the definitions:

$$\mathsf{L} \subseteq \mathsf{NL}$$
$$\mathsf{P} \subseteq \mathsf{NP}$$
$$\mathsf{PSPACE} \subseteq \mathsf{NPSPACE}$$

$\mathsf{NP} \subseteq \mathsf{PSPACE}$

To simulate a nondeterministic machine M running in time $t(n)$ by a deterministic one, it suffices to carry out a *depth-first* search of the computation tree.

We keep a counter to cut off branches that exceed $t(n)$ steps.

The space required is:

- a *counter* to count up to $t(n)$; and
- a *stack* of configurations, each of size at most $O(t(n))$.

The depth of the stack is at most $t(n)$.

Thus, if t is a polynomial, the total space required is polynomial.

$\mathsf{NL} \subseteq \mathsf{P}$

Given a nondeterministic machine M that works with *work space* bounded by $s(n)$ and an input x of length n, there is some constant c such that

> the total number of possible configurations of M within space bounds $s(n)$ is bounded by $n \cdot c^{s(n)}$.

Define the *configuration graph* of M, x to be the graph whose nodes are the possible configurations, and there is an edge from i to j if, and only if, $i \rightarrow_M j$.

Reachability in the Configuration Graph

M accepts x if, and only if, some accepting configuration is reachable from the starting configuration in the configuration graph of M, x.

Using an $O(n^2)$ algorithm for $\mathsf{Reachability}$, we get that M can be simulated by a deterministic machine operating in time

$$c'(nc^{s(n)})^2 \sim c'c^{2(\log n + s(n))} \sim d^{(\log n + s(n))}$$

for some constant d.

When $s(n) = O(\log n)$, this is polynomial and so $\mathsf{NL} \subseteq \mathsf{P}$.

When $s(n)$ is polynomial this is exponential in n and so $\mathsf{NPSPACE} \subseteq \mathsf{EXP}$.

Nondeterministic Space Classes

If *Reachability* were solvable by a *deterministic* machine with logarithmic space, then

$$\mathsf{L} = \mathsf{NL}.$$

In fact, *Reachability* is solvable by a deterministic machine with space $O((\log n)^2)$.

This implies

$$\mathsf{NSPACE}(s(n)) \subseteq \mathsf{SPACE}((s(n)^2)).$$

In particular PSPACE = NPSPACE.

Reachability in $O((\log n)^2)$

$O((\log n)^2)$ space Reachability algorithm:

```
Path(a, b, i)
if $i = 1$ and $(a, b)$ is not an edge reject
else if $(a, b)$ is an edge or $a = b$ accept
else, for each node $x$, check:
  1. is there a path $a - x$ of length $i/2$; and
  2. is there a path $x - b$ of length $i/2$?
if such an $x$ is found, then accept, else reject.
```

The maximum depth of recursion is $\log n$, and the number of bits of information kept at each stage is $3 \log n$.

Inclusions between Classes

This leaves us with the following:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$$

Hierarchy Theorems proved by *diagonalization* can show that:

$$\mathsf{L} \neq \mathsf{PSPACE} \quad \mathsf{NL} \neq \mathsf{NPSPACE} \quad \mathsf{P} \neq \mathsf{EXP}$$

For other inclusions above, it remains an open question whether they are strict.

Complement Classes

If we interchange accepting and rejecting states in a deterministic machine that accepts the language L, we get one that accepts $\overline{L}$.

If a language $L \in \mathsf{P}$, then also $\overline{L} \in \mathsf{P}$.

Complexity classes defined in terms of nondeterministic machine models are not necessarily closed under complementation of languages.

Define,

co-NP – the languages whose complements are in NP.

co-NL – the languages whose complements are in NL.

Relationships

P $\subseteq$ NP $\cap$ co-NP and any of the situations is consistent with our present state of knowledge:

- P $=$ NP $=$ co-NP
- P $=$ NP $\cap$ co-NP $\neq$ NP $\neq$ co-NP
- P $\neq$ NP $\cap$ co-NP $=$ NP $=$ co-NP
- P $\neq$ NP $\cap$ co-NP $\neq$ NP $\neq$ co-NP

It follows from the fact that PSPACE $=$ NPSPACE that NPSPACE is closed under complementation.

Also, **Immerman and Szelepcsényi** showed that NL $=$ co-NL.

Reductions

Given two languages $L_1 \subseteq \Sigma_1^\star$, and $L_2 \subseteq \Sigma_2^\star$,

a *reduction* of L_1 to L_2 is a *computable* function

$$f : \Sigma_1^\star \to \Sigma_2^\star$$

such that for every string $x \in \Sigma_1^\star$,

$$f(x) \in L_2 \text{ if, and only if, } x \in L_1$$

Resource Bounded Reductions

If f is computable by a polynomial time algorithm, we say that L_1 is *polynomial time reducible* to L_2.

$$L_1 \leq_P L_2$$

If f is also computable in SPACE$(\log n)$, we write

$$L_1 \leq_L L_2$$

Reductions 2

If $L_1 \leq L_2$ we understand that L_1 is no more difficult to solve than L_2.

That is to say, for any of the complexity classes $\mathcal{C}$ we consider,

If $L_1 \leq L_2$ and $L_2 \in \mathcal{C}$, then $L_1 \in \mathcal{C}$

We can get an algorithm to decide L_1 by first computing f, and then using the $\mathcal{C}$-algorithm for L_2.

Provided that $\mathcal{C}$ is *closed* under such reductions.

Completeness

The usefulness of reductions is that they allow us to establish the *relative* complexity of problems, even when we cannot prove absolute lower bounds.

Cook (1972) first showed that there are problems in NP that are maximally difficult.

For any complexity class $\mathcal{C}$, a language L is said to be *$\mathcal{C}$-hard* if for every language $A \in \mathcal{C}$, $A \leq L$.

A language L is *$\mathcal{C}$-complete* if it is in $\mathcal{C}$ and it is $\mathcal{C}$-hard.

Complete Problems

Examples of complete problems for various complexity classes.

NL
Reachability

P
Game, Circuit Value Problem

NP Satisfiability of Boolean Formulas, Graph 3-Colourability, Hamiltonian Cycle

co-NP
Validity of Boolean Formulas, Non 3-colourability

PSPACE
Geography, The game of HEX

Reading List for this Part

1. Papadimitriou. Chapters 7, 8 and 16.
2. Arora and Barak. Chapters 2 and 4.
3. Immerman Chapter 2.

Topics in Logic and Complexity
Part 3

Anuj Dawar

MPhil Advanced Computer Science, Lent 2013

P-complete Problems

Game

Input: A directed graph $G = (V, E)$ with a partition $V = V_1 \cup V_2$ of the vertices and two distinguished vertices $s, t \in V$.

Decide: whether Player 1 can force a token from s to t in the game where when the token is on $v \in V_1$, Player 1 moves it along an edge leaving v and when it is on $v \in V_2$, Player 2 moves it along an edge leaving v.

Circuit Value Problem

A *Circuit* is a *directed acyclic graph* $G = (V, E)$ where each node has *in-degree* $0, 1$ or 2 and there is exactly one vertex t with no outgoing edges, along with a labelling which assigns:

- to each node of indegree 0 a value of 0 or 1
- to each node of indegree 1 a label $\neg$
- to each node of indegree 2 a label $\wedge$ or $\vee$

The problem CVP is, given a circuit, decide if the target node t evaluates to 1.

NP-complete Problems

SAT

Input: A Boolean formula ϕ

Decide: if there is an assignment of truth values to the variables of ϕ that makes ϕ true.

Hamiltonicity

Input: A graph $G = (V, E)$

Decide: if there is a cycle in G that visits every vertex exactly once.

co-NP-complete Problems

VAL

Input: A Boolean formula ϕ

Decide: if every assignment of truth values to the variables of ϕ makes ϕ true.

Non-3-colourability

Input: A graph $G = (V, E)$

Decide: if there is no function $\chi : V \to \{1, 2, 3\}$ such that the two endpoints of every edge are differently coloured.

PSPACE-complete Problems

Geography is very much like *Game* but now players are not allowed to visit a vertex that has been previously visitied.

HEX is a game played by two players on a graph $G = (V, E)$ with a source and target $s, t \in V$.

The two players take turns selecting vertices from V—neither player can choose a vertex that has been previously selected. Player 1 wins if, at any point, the vertices she has selected include a path from s to t. Player 2 wins if all vertices have been selected and no such path is formed.

The problem is to decide which player has a winning strategy.

Descriptive Complexity

Descriptive Complexity provides an alternative perspective on Computational Complexity.

Computational Complexity

- Measure use of resources (space, time, *etc.*) on a machine model of computation;
- Complexity of a language—i.e. a set of strings.

Descriptive Complexity

- Complexity of a class of structures—e.g. a collection of graphs.
- Measure the complexity of describing the collection in a formal logic, using resources such as variables, quantifiers, higher-order operators, *etc.*

There is a fascinating interplay between the views.

Signature and Structure

In general a *signature* (or *vocabulary*) σ is a finite sequence of *relation*, *function* and *constant* symbols:

$$\sigma = (R_1, \ldots, R_m, f_1, \ldots, f_n, c_1, \ldots, c_p)$$

where, associated with each relation and function symbol is an arity.

Structure

A structure $\mathbb{A}$ over the signature σ is a tuple:

$$\mathbb{A} = (A, R_1^{\mathbb{A}}, \ldots, R_m^{\mathbb{A}}, f_1^{\mathbb{A}}, \ldots, f_n^{\mathbb{A}}, c_1^{\mathbb{A}}, \ldots, c_n^{\mathbb{A}}),$$

where,

- A is a non-empty set, the *universe* of the strucure $\mathbb{A}$,
- each $R_i^{\mathbb{A}}$ is a relation over A of the appropriate arity.
- each $f_i^{\mathbb{A}}$ is a function over A of the appropriate arity.
- each $c_i^{\mathbb{A}}$ is an element of A.

First-order Logic

Formulas of *first-order logic* are formed from the signature σ and an infinite collection X of variables as follows.

terms – c, x, $f(t_1, \ldots, t_a)$

Formulas are defined by induction:

- *atomic formulas* – $R(t_1, \ldots, t_a)$, $t_1 = t_2$
- *Boolean operations* – $\phi \wedge \psi$, $\phi \vee \psi$, $\neg\phi$
- *first-order quantifiers* – $\exists x\phi$, $\forall x\phi$

Queries

A formula ϕ with free variables among $x_1, \ldots, x_n$ defines a map Q from structures to relations:

$$Q(\mathbb{A}) = \{\mathbf{a} \mid \mathbb{A} \models \phi[\mathbf{a}]\}.$$

Any such map Q which associates to every structure $\mathbb{A}$ a (n-ary) relation on A, and is isomorphism invariant, is called a *(n-ary) query*.

Q is *isomorphism invariant* if, whenever $f : A \to B$ is an isomorphism between $\mathbb{A}$ and $\mathbb{B}$, it is also an isomorphism between $(A, Q(\mathbb{A}))$ and $(B, Q(\mathbb{B}))$.

If $n = 0$, we can regard the query as a map from structures to $\{0, 1\}$—a *Boolean query*.

Graphs

For example, take the signature (E), where E is a binary relation symbol.

Finite structures (V, E) of this signature are directed graphs.

Moreover, the class of such finite structures satisfying the sentence

$$\forall x \neg Exx \wedge \forall x \forall y (Exy \to Eyx)$$

can be identified with the class of (*loop-free, undirected*) graphs.

Complexity

For a first-order sentence ϕ, we ask what is the *computational complexity* of the problem:

Input: a structure $\mathbb{A}$
Decide: if $\mathbb{A} \models \phi$

In other words, how complex can the collection of finite models of ϕ be?

In order to talk of the complexity of a class of finite structures, we need to fix some way of representing finite structures as strings.

Representing Structures as Strings

We use an alphabet $\Sigma = \{0, 1, \#, -\}$.

For a structure $\mathbb{A} = (A, R_1, \ldots, R_m, f_1, \ldots, f_l)$, fix a linear order $<$ on $A = \{a_1, \ldots, a_n\}$.

R_i (of arity k) is encoded by a string $[R_i]_<$ of 0s and 1s of length n^k.

f_i is encoded by a string $[f_i]_<$ of 0s, 1s and $-$s of length $n^k \log n$.

$$[\mathbb{A}]_< = \underbrace{1 \cdots 1}_{n} \# [R_1]_< \# \cdots \# [R_m]_< \# [f_1]_< \# \cdots \# [f_l]_<$$

The exact string obtained depends on the choice of order.

Naïve Algorithm

The straightforward algorithm proceeds recursively on the structure of ϕ:

- Atomic formulas by direct lookup.
- Boolean connectives are easy.
- If $\phi \equiv \exists x\, \psi$ then for each $a \in \mathbb{A}$ check whether

$$(\mathbb{A}, c \mapsto a) \models \psi[c/x],$$

where c is a new constant symbol.

This runs in time $O(ln^m)$ and $O(m \log n)$ space, where m is the nesting depth of quantifiers in ϕ.

$$\mathrm{Mod}(\phi) = \{\mathbb{A} \mid \mathbb{A} \models \phi\}$$

is in *logarithmic space* and *polynomial time*.

Reading List for this Part

1. Papadimitriou. Chapter 8.
2. Libkin Chapter 2.
3. Grädel et al. Sections 2.1–2.4 (Kolaitis).

Topics in Logic and Complexity Part 4

Anuj Dawar

MPhil Advanced Computer Science, Lent 2013

Complexity of First-Order Logic

The following problem:

FO satisfaction

> *Input*: a structure $\mathbb{A}$ and a first-order sentence ϕ
> *Decide*: if $\mathbb{A} \models \phi$

is PSPACE-complete.

It follows from the $O(ln^m)$ and $O(m \log n)$ space algorithm that the problem is in PSPACE.

How do we prove completeness?

QBF

We define *quantified Boolean formulas* inductively as follows, from a set $\mathcal{X}$ of *propositional variables*.

- A propositional constant T or F is a formula
- A propositional variable $X \in \mathcal{X}$ is a formula
- If ϕ and ψ are formulas then so are: $\neg\phi$, $\phi \wedge \psi$ and $\phi \vee \psi$
- If ϕ is a formula and X is a variable then $\exists X\ \phi$ and $\forall X\ \phi$ are formulas.

Say that an occurrence of a variable X is *free* in a formula ϕ if it is not within the scope of a quantifier of the form $\exists X$ or $\forall X$.

QBF

Given a quantified Boolean formula ϕ and an assignment of *truth values* to its free variables, we can ask whether ϕ evaluates to *true* or *false*.

In particular, if ϕ has no free variables, then it is equivalent to either *true* or *false*.

QBF is the following decision problem:

> *Input:* a quantified Boolean formula ϕ with no free variables.
> *Decide:* whether ϕ evaluates to *true*.

Complexity of QBF

Note that a Boolean formula ϕ *without quantifiers* and with variables $X_1, \ldots, X_n$ is *satisfiable* if, and only if, the formula

$$\exists X_1 \cdots \exists X_n\ \phi \quad \text{is } \textit{true}.$$

Similarly, ϕ is *valid* if, and only if, the formula

$$\forall X_1 \cdots \forall X_n\ \phi \quad \text{is } \textit{true}.$$

Thus, SAT $\leq_L$ QBF and VAL $\leq_L$ QBF and so QBF is NP-hard and co-NP-hard.

In fact, QBF is PSPACE-complete.

QBF is in PSPACE

To see that QBF is in PSPACE, consider the algorithm that maintains a 1-bit register X for each Boolean variable appearing in the input formula ϕ and evaluates ϕ in the natural fashion.

The crucial cases are:

- If ϕ is $\exists X\ \psi$ then return T if *either* $(X \leftarrow \mathsf{T} \quad ; \quad \mathsf{evaluate}\ \psi)$ *or* $(X \leftarrow \mathsf{F} \quad ; \quad \mathsf{evaluate}\ \psi)$ returns T.
- If ϕ is $\forall X\ \psi$ then return T if *both* $(X \leftarrow \mathsf{T} \quad ; \quad \mathsf{evaluate}\ \psi)$ *and* $(X \leftarrow \mathsf{F} \quad ; \quad \mathsf{evaluate}\ \psi)$ return T.

PSPACE-completeness

To prove that QBF is PSPACE-complete, we want to show:

> Given a machine M with a polynomial space bound and an input x, we can define a quantified Boolean formula ϕ_x^M which evaluates to *true* if, and only if, M accepts x.
>
> Moreover, ϕ_x^M can be computed from x in *polynomial time* (or even *logarithmic space*).

The number of distinct configurations of M on input x is bounded by 2^{n^k} for some k $(n = |x|)$.

Each configuration can be represented by n^k bits.

Constructing ϕ_x^M

We use tuples $\mathbf{A}, \mathbf{B}$ of n^k Boolean variables each to encode *configurations* of M.

Inductively, we define a formula $\psi_i(\mathbf{A}, \mathbf{B})$ which is *true* if the configuration coded by $\mathbf{B}$ is reachable from that coded by $\mathbf{A}$ in at most 2^i steps.

$$\begin{aligned}
\psi_0(\mathbf{A}, \mathbf{B}) &\equiv \text{“}\mathbf{A} = \mathbf{B}\text{”} \vee \text{“}\mathbf{A} \rightarrow_M \mathbf{B}\text{”} \\
\psi_{i+1}(\mathbf{A}, \mathbf{B}) &\equiv \exists \mathbf{Z} \forall \mathbf{X} \forall \mathbf{Y}\ [(\mathbf{X} = \mathbf{A} \wedge \mathbf{Y} = \mathbf{Z}) \vee (\mathbf{X} = \mathbf{Z} \wedge \mathbf{Y} = \mathbf{B}) \\
&\qquad \Rightarrow \psi_i(\mathbf{X}, \mathbf{Y})] \\
\phi &\equiv \psi_{n^k}(\mathbf{A}, \mathbf{B}) \wedge \text{“}\mathbf{A} = \mathsf{start}\text{”} \wedge \text{“}\mathbf{B} = \mathsf{accept}\text{”}
\end{aligned}$$

Reducing QBF to FO satisfaction

We have seen that *FO satisfaction* is in PSPACE.

To show that it is PSPACE-complete, it suffices to show that QBF $\leq_L$ FO sat.

The reduction maps a quantified Boolean formula ϕ to a pair $(\mathbb{A}, \phi^*)$ where $\mathbb{A}$ is a structure with two elements: 0 and 1 interpreting two constants f and t respectively.

ϕ^* is obtained from ϕ by a simple inductive definition.

Expressive Power of FO

For any *fixed* sentence ϕ of first-order logic, the class of structures $\mathrm{Mod}(\phi)$ is in L.

There are computationally easy properties that are not definable in first-order logic.

- There is no sentence ϕ of first-order logic such that $\mathbb{A} \models \phi$ if, and only if, $|A|$ is even.
- There is no formula $\phi(E, x, y)$ that defines the transitive closure of a binary relation E.

We will see proofs of these facts later on.

Second-Order Logic

We extend first-order logic by a set of *relational variables*.

For each $m \in \mathbb{N}$ there is an infinite collection of variables $\mathcal{V}^m = \{V_1^m, V_2^m, \ldots\}$ of *arity* m.

Second-order logic extends first-order logic by allowing *second-order quantifiers*

$$\exists X\, \phi \quad \text{for } X \in \mathcal{V}^m$$

A structure $\mathbb{A}$ satisfies $\exists X\, \phi$ if there is an m-ary relation R on the universe of $\mathbb{A}$ such that $(\mathbb{A}, X \to R)$ satisfies ϕ.

Existential Second-Order Logic

ESO—*existential second-order logic* consists of those formulas of second-order logic of the form:

$$\exists X_1 \cdots \exists X_k\, \phi$$

where ϕ is a first-order formula.

Examples

Evennness

This formula is true in a structure if, and only if, the size of the domain is even.

$$\begin{aligned}
\exists B \exists S \quad & \forall x \exists y B(x,y) \land \forall x \forall y \forall z B(x,y) \land B(x,z) \to y = z \\
& \forall x \forall y \forall z B(x,z) \land B(y,z) \to x = y \\
& \forall x \forall y S(x) \land B(x,y) \to \neg S(y) \\
& \forall x \forall y \neg S(x) \land B(x,y) \to S(y)
\end{aligned}$$

Examples

Transitive Closure

This formula is true of a pair of elements a, b in a structure if, and only if, there is an E-path from a to b.

$$\begin{aligned}\exists P \quad & \forall x \forall y\, P(x,y) \to E(x,y)\\ & \exists x P(a,x) \land \exists x P(x,b) \land \neg \exists x P(x,a) \land \neg \exists x P(b,x)\\ & \forall x \forall y (P(x,y) \to \forall z (P(x,z) \to y = z))\\ & \forall x \forall y (P(x,y) \to \forall z (P(z,y) \to x = z))\\ & \forall x ((x \neq a \land \exists y P(x,y)) \to \exists z P(z,x))\\ & \forall x ((x \neq b \land \exists y P(y,x)) \to \exists z P(x,z))\end{aligned}$$

Examples

3-Colourability

The following formula is true in a graph (V, E) if, and only if, it is 3-colourable.

$$\begin{aligned}\exists R \exists B \exists G \quad & \forall x (Rx \lor Bx \lor Gx) \land\\ & \quad \forall x (\quad \neg(Rx \land Bx) \land \neg(Bx \land Gx) \land \neg(Rx \land Gx)) \land\\ & \quad \forall x \forall y (Exy \to (\quad \neg(Rx \land Ry) \land\\ & \qquad\qquad\qquad\qquad \neg(Bx \land By) \land\\ & \qquad\qquad\qquad\qquad \neg(Gx \land Gy)))\end{aligned}$$

Reading List for this Part

1. Papadimitriou. Chapter 5. Section 19.1.
2. Grädel et al. Section 3.1

Topics in Logic and Complexity Part 5

Anuj Dawar

MPhil Advanced Computer Science, Lent 2013

Fagin's Theorem

Theorem (Fagin)
A class $\mathcal{C}$ of finite structures is definable by a sentence of *existential second-order logic* if, and only if, it is decidable by a *nondeterminisitic machine* running in polynomial time.

$$\mathsf{ESO} = \mathsf{NP}$$

One direction is easy: Given $\mathbb{A}$ and $\exists P_1 \ldots \exists P_m \phi$.

a nondeterministic machine can guess an interpretation for $P_1, \ldots, P_m$ and then verify ϕ.

Fagin's Theorem

Given a machine M and an integer k, there is an ESO sentence ϕ such that $\mathbb{A} \models \phi$ if, and only if, M accepts $[\mathbb{A}]_<$, for some order $<$ in n^k steps.

We construct a *first-order* formula $\phi_{M,k}$ such that

$$(\mathbb{A}, <, \mathbf{X}) \models \phi_{M,k} \quad \Leftrightarrow \quad \begin{array}{l}\mathbf{X} \text{ codes an accepting computation of } M \\ \text{of length at most } n^k \text{ on input } [\mathbb{A}]_<\end{array}$$

So, $\mathbb{A} \models \exists < \exists \mathbf{X}\, \phi_{M,k}$ if, and only if, there is some order $<$ on $\mathbb{A}$ so that M accepts $[\mathbb{A}]_<$ in time n^k.

Order

The formula $\phi_{M,k}$ is built up as the *conjunction* of a number of formulas. The first of these simply says that $<$ is a *linear order*

$$\begin{array}{l}\forall x(\neg x < x) \wedge \\ \forall x \forall y(x < y \rightarrow \neg y < x) \wedge \\ \forall x \forall y(x < y \vee y < x \vee x = y) \\ \forall x \forall y \forall z(x < y \wedge y < z \rightarrow x < z)\end{array}$$

We can use a linear order on the elements of $\mathbb{A}$ to define a lexicographic order on k-tuples.

Ordering Tuples

If $\mathbf{x} = x_1, \ldots, x_k$ and $\mathbf{y} = y_1, \ldots, y_k$ are k-tuples of variables, we use $\mathbf{x} = \mathbf{y}$ as shorthand for the formula $\bigvee_{i<k} x_i = y_i$ and $\mathbf{x} < \mathbf{y}$ as shorthand for the formula

$$\bigvee_{i<k} ((\bigvee_{j<i} x_j = y_j) \wedge x_i < y_i)$$

We also write $\mathbf{y} = \mathbf{x} + 1$ for the following formula:

$$\mathbf{x} < \mathbf{y} \wedge \forall \mathbf{z}\big(\mathbf{x} < \mathbf{z} \rightarrow (\mathbf{y} = \mathbf{z} \vee \mathbf{y} < \mathbf{z})\big)$$

Constructing the Formula

Let $M = (K, \Sigma, s, \delta)$.

The tuple $\mathbf{X}$ of second-order variables appearing in $\phi_{M,k}$ contains the following:

S_q	a k-ary relation symbol for each $q \in K$
T_σ	a $2k$-ary relation symbol for each $\sigma \in \Sigma$
H	a $2k$-ary relation symbol

Intuitively, these relations are intended to capture the following:

- $S_q(\mathbf{x})$ – the state of the machine at time $\mathbf{x}$ is q.
- $T_\sigma(\mathbf{x}, \mathbf{y})$ – at time $\mathbf{x}$, the symbol at position $\mathbf{y}$ of the tape is σ.
- $H(\mathbf{x}, \mathbf{y})$ – at time $\mathbf{x}$, the tape head is pointing at tape cell $\mathbf{y}$.

We now have to see how to write the formula $\phi_{M,k}$, so that it enforces these meanings.

Initial state is s and the head is initially at the beginning of the tape.

$$\forall \mathbf{x} \big((\forall \mathbf{y}\ \mathbf{x} \leq \mathbf{y}) \rightarrow S_s(\mathbf{x}) \wedge H(\mathbf{x}, \mathbf{x}) \big)$$

The head is never in two places at once

$$\forall \mathbf{x} \forall \mathbf{y} \big(H(\mathbf{x}, \mathbf{y}) \rightarrow (\forall \mathbf{z} (\mathbf{y} \neq \mathbf{z}) \rightarrow (\neg H(\mathbf{x}, \mathbf{z}))) \big)$$

The machine is never in two states at once

$$\forall \mathbf{x} \bigwedge_q (S_q(\mathbf{x}) \rightarrow \bigwedge_{q' \neq q} (\neg S_{q'}(\mathbf{x})))$$

Each tape cell contains only one symbol

$$\forall \mathbf{x} \forall \mathbf{y} \bigwedge_\sigma (T_\sigma(\mathbf{x}, \mathbf{y}) \rightarrow \bigwedge_{\sigma' \neq \sigma} (\neg T_{\sigma'}(\mathbf{x}, \mathbf{y})))$$

Initial Tape Contents

The initial contents of the tape are $[\mathbb{A}]_<$.

$$\begin{array}{ll} \forall \mathbf{x} & \mathbf{x} \leq n \rightarrow T_1(\mathbf{1}, \mathbf{x}) \wedge \\ & \mathbf{x} \leq n^a \rightarrow (T_1(\mathbf{1}, \mathbf{x} + n + 1) \leftrightarrow R_1(\mathbf{x}|_a)) \\ \ldots & \end{array}$$

where,

$$\mathbf{x} < n^a \quad : \quad \bigwedge_{i \leq (k-a)} x_i = 0$$

The tape does not change except under the head

$$\forall \mathbf{x} \forall \mathbf{y} \forall \mathbf{z} (\mathbf{y} \neq \mathbf{z} \rightarrow (\bigwedge_{\sigma} (H(\mathbf{x}, \mathbf{y}) \wedge T_{\sigma}(\mathbf{x}, \mathbf{z}) \rightarrow T_{\sigma}(\mathbf{x}+1, \mathbf{z})))$$

Each step is according to δ.

$$\forall \mathbf{x} \forall \mathbf{y} \bigwedge_{\sigma} \bigwedge_{q} (H(\mathbf{x}, \mathbf{y}) \wedge S_q(\mathbf{x}) \wedge T_{\sigma}(\mathbf{x}, \mathbf{y})) \\ \rightarrow \bigvee_{\Delta} (H(\mathbf{x}+1, \mathbf{y}') \wedge S_{q'}(\mathbf{x}+1) \wedge T_{\sigma'}(\mathbf{x}+1, \mathbf{y}))$$

where Δ is the set of all triples (q', σ', D) such that $((q, \sigma), (q', \sigma', D)) \in \delta$ and

$$j' = \begin{cases} j & \text{if } D = S \\ j-1 & \text{if } D = L \\ j+1 & \text{if } D = R \end{cases}$$

Finally, some accepting state is reached

$$\exists \mathbf{x}\, S_{\text{acc}}(\mathbf{x})$$

NP

Recall that a languge L is in NP if, and only if,

$$L = \{x \mid \exists y R(x, y)\}$$

where R is *polynomial-time decidable* and *polynomially-balanced*.

Fagin's theorem tells us that polynomial-time decidability can, in some sense, be replaced by *first-order definability*.

co-NP

USO—*universal second-order logic* consists of those formulas of second-order logic of the form:

$$\forall X_1 \cdots \forall X_k\, \phi$$

where ϕ is a first-order formula.

A corollary of Fagin's theorem is that a class $\mathcal{C}$ of finite structures is definable by a sentence of *existential second-order logic* if, and only if, it is decidable by a *nondeterminisitic machine* running in polynomial time.

$$\text{USO} = \text{co-NP}$$

Second-Order Alternation Hierarchy

We can define further classes by allowing other second-order *quantifier prefixes*.

$\Sigma^1_1 =$ ESO

$\Pi^1_1 =$ USO

Σ^1_{n+1} is the collection of properties definable by a sentence of the form: $\exists X_1 \cdots \exists X_k\, \phi$ where ϕ is a Π^1_n formula.

Π^1_{n+1} is the collection of properties definable by a sentence of the form: $\forall X_1 \cdots \forall X_k\, \phi$ where ϕ is a Σ^1_n formula.

Note: every formula of second-order logic is Σ^1_n and Π^1_n for some n.

Polynomial Hierarchy

We have, for each n:

$$\Sigma^1_n \cup \Pi^1_n \quad \subseteq \quad \Sigma^1_{n+1} \cap \Pi^1_{n+1}$$

The classes together form the *polynomial hierarchy* or PH.

NP $\subseteq$ PH $\subseteq$ PSPACE

P $=$ NP if, and only if, P $=$ PH

Reading List for this Part

1. Arora and Barak Chapter 5.
2. Grädel et al. Section 3.2
3. Libkin. Chapter 9.
4. Ebbinghaus and Flum. Chapter 7.

Topics in Logic and Complexity
Part 6

Anuj Dawar

MPhil Advanced Computer Science, Lent 2013

Expressive Power of First-Order Logic

We noted that there are computationally easy properties that are not definable in first-order logic.

- There is no sentence ϕ of first-order logic such that $\mathbb{A} \models \phi$ if, and only if, $|A|$ is even.
- There is no sentence ϕ that defines exactly the *connected* graphs.

How do we *prove* these facts?

Our next aim is to develop the tools that enable such proofs.

Quantifier Rank

The *quantifier rank* of a formula ϕ, written $\text{qr}(\phi)$ is defined inductively as follows:

1. if ϕ is atomic then $\text{qr}(\phi) = 0$,
2. if $\phi = \neg\psi$ then $\text{qr}(\phi) = \text{qr}(\psi)$,
3. if $\phi = \psi_1 \vee \psi_2$ or $\phi = \psi_1 \wedge \psi_2$ then $\text{qr}(\phi) = \max(\text{qr}(\psi_1), \text{qr}(\psi_2))$.
4. if $\phi = \exists x\psi$ or $\phi = \forall x\psi$ then $\text{qr}(\phi) = \text{qr}(\psi) + 1$

More informally, $\text{qr}(\phi)$ is the *maximum depth of nesting of quantifiers* inside ϕ.

Formulas of Bounded Quantifier Rank

Note: For the rest of this lecture, we assume that our signature consists only of relation and constant symbols. That is, there are *no function symbols of non-zero arity*.

With this proviso, it is easily proved that in a finite vocabulary, for each q, there are (up to logical equivalence) only finitely many sentences ϕ with $\text{qr}(\phi) \leq q$.

To be precise, we prove by induction on q that for all m, there are only finitely many formulas of quantifier rank q with at most m free variables.

Formulas of Bounded Quantifier Rank

If $\text{qr}(\phi) = 0$ then ϕ is a Boolean combination of atomic formulas. If it is has m variables, it is equivalent to a formula using the variables $x_1, \ldots, x_m$. There are finitely many formulas, *up to logical equivalence*.

Suppose $\text{qr}(\phi) = q + 1$ and the *free variables* of ϕ are among $x_1, \ldots, x_m$. Then ϕ is a Boolean combination of formulas of the form

$$\exists x_{m+1}\psi$$

where ψ is a formula with $\text{qr}(\psi) = q$ and free variables $x_1, \ldots, x_m, x_{m+1}$.

By induction hypothesis, there are only finitely many such formulas, and therefore finitely many Boolean combinations.

Equivalence Relation

For two structures $\mathbb{A}$ and $\mathbb{B}$, we say $\mathbb{A} \equiv_q \mathbb{B}$ if for any sentence ϕ with $\mathrm{qr}(\phi) \leq q$,

$$\mathbb{A} \models \phi \text{ if, and only if, } \mathbb{B} \models \phi.$$

More generally, if $\mathbf{a}$ and $\mathbf{b}$ are m-tuples of elements from $\mathbb{A}$ and $\mathbb{B}$ respectively, then we write $(\mathbb{A}, \mathbf{a}) \equiv_q (\mathbb{B}, \mathbf{b})$ if for any formula ϕ with m free variables $\mathrm{qr}(\phi) \leq q$,

$$\mathbb{A} \models \phi[\mathbf{a}] \text{ if, and only if, } \mathbb{B} \models \phi[\mathbf{b}].$$

Partial Isomorphisms

A map f is a partial isomorphism between structures $\mathbb{A}$ and $\mathbb{B}$, if

- the domain of $f = \{a_1, \ldots, a_l\} \subseteq A$, including the interpretation of all constants;
- the range of $f = \{b_1, \ldots, b_l\} \subseteq B$, including the interpretation of all constants; and
- f is an isomorphism between its domain and range.

Note that if f is a partial isomorphism taking a tuple $\mathbf{a}$ to a tuple $\mathbf{b}$, then for any *quantifier-free* formula θ

$$\mathbb{A} \models \theta[\mathbf{a}] \text{ if, and only if, } \mathbb{B} \models \theta[\mathbf{b}].$$

Ehrenfeucht-Fraïssé Games

The q-round Ehrenfeucht game on structures $\mathbb{A}$ and $\mathbb{B}$ proceeds as follows:

- There are two players called Spoiler and Duplicator.
- At the ith round, Spoiler chooses one of the structures (say $\mathbb{B}$) and one of the elements of that structure (say b_i).
- Duplicator must respond with an element of the other structure (say a_i).
- If, after q rounds, the map $a_i \mapsto b_i$ is a partial isomorphism, then Duplicator has won the game, otherwise Spoiler has won.

Equivalence and Games

Write $\mathbb{A} \sim_q \mathbb{B}$ to denote the fact that *Duplicator* has a *winning strategy* in the q-round Ehrenfeucht game on $\mathbb{A}$ and $\mathbb{B}$.

The relation $\sim_q$ is, in fact, an *equivalence relation*.

Theorem (Fraïssé 1954; Ehrenfeucht 1961)
$\mathbb{A} \sim_q \mathbb{B}$ if, and only if, $\mathbb{A} \equiv_q \mathbb{B}$

While one direction $\mathbb{A} \sim_q \mathbb{B} \Rightarrow \mathbb{A} \equiv_q \mathbb{B}$ is true for an arbitrary vocabulary, the other direction assumes that the vocabulary is *finite* and has *no function symbols*.

Proof

To prove $\mathbb{A} \sim_q \mathbb{B} \Rightarrow \mathbb{A} \equiv_q \mathbb{B}$, it suffices to show that if there is a sentence ϕ with $\mathrm{qr}(\phi) \leq q$ such that

$$\mathbb{A} \models \phi \quad \text{and} \quad \mathbb{B} \not\models \phi$$

then *Spoiler* has a winning strategy in the q-round Ehrenfeucht game on $\mathbb{A}$ and $\mathbb{B}$.

Assume that ϕ is in *negation normal form*, i.e. all negations are in front of atomic formulas.

Proof

We prove by induction on q the stronger statement that if ϕ is a formula with $\mathrm{qr}(\phi) \leq q$ and $\mathbf{a} = (a_1, \ldots, a_m)$ and $\mathbf{b} = (b_1, \ldots, b_m)$ are tuples of elements from $\mathbb{A}$ and $\mathbb{B}$ respectively such that

$$\mathbb{A} \models \phi[\mathbf{a}] \quad \text{and} \quad \mathbb{B} \not\models \phi[\mathbf{b}]$$

then *Spoiler* has a winning strategy in the q-round Ehrenfeucht game which starts from a position in which $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$ have *already been selected*.

Proof

When $q = 0$, ϕ is a quantifier-free formula. Thus, if

$$\mathbb{A} \models \phi[\mathbf{a}] \quad \text{and} \quad \mathbb{B} \not\models \phi[\mathbf{b}]$$

there is an *atomic* formula θ that distinguishes the two tuples and therefore the map taking $\mathbf{a}$ to $\mathbf{b}$ is not a *partial isomorphism*.

When $q = p + 1$, there is a subformula θ of ϕ of the form $\exists x\psi$ or $\forall x\psi$ such that $\mathrm{qr}(\psi) \leq p$ and

$$\mathbb{A} \models \theta[\mathbf{a}] \quad \text{and} \quad \mathbb{B} \not\models \theta[\mathbf{b}]$$

If $\theta = \exists x\psi$, *Spoiler* chooses a witness for x in $\mathbb{A}$.

If $\theta = \forall x\psi$, $\mathbb{B} \models \exists x\neg\psi$ and *Spoiler* chooses a witness for x in $\mathbb{B}$.

Using Games

To show that a class of structures S is not definable in FO, we find, for every q, a pair of structures $\mathbb{A}_q$ and $\mathbb{B}_q$ such that

- $\mathbb{A}_q \in S$, $\mathbb{B}_q \in \overline{S}$; and
- *Duplicator* wins a q-round game on $\mathbb{A}_q$ and $\mathbb{B}_q$.

This shows that S is not closed under the relation $\equiv_q$ for *any* q.

Fact:

> S is definable by a first order sentence if, and only if, S is closed under the relation $\equiv_q$ for some q.

The direction from right to left requires a *finite, function-free* vocabulary.

Evenness

Let $\mathbb{A}$ be a structure in the *empty vocabulary* with q elements and $\mathbb{B}$ be a structure with $q+1$ elements.

Then, it is easy to see that $\mathbb{A} \sim_q \mathbb{B}$.

It follows that there is no first-order sentence that defines the structures with an even number of elements.

If $S \subseteq \mathbb{N}$ is a set such that

$$\{\mathbb{A} \mid |\mathbb{A}| \in S\}$$

is definable by a first-order sentence then S is finite or co-finite.

Linear Orders

Let L_n denote the structure in one binary relation $\leq$ which is a linear order of n elements. Then $L_6 \not\equiv_3 L_7$ but $L_7 \equiv_3 L_8$.

In general, for $m, n \geq 2^p - 1$,

$$L_m \equiv_p L_n$$

Duplicator's strategy is to maintain the following condition after r rounds of the game:

for $1 \leq i < j \leq r$,

- *either* $\text{length}(a_i, a_j) = \text{length}(b_i, b_j)$
- *or* $\text{length}(a_i, a_j), \text{length}(b_i, b_j) \geq 2^{p-r} - 1$

Evenness is not first order definable, even on linear orders.

Reading List for this Part

1. Ebbinghaus and Flum. Chapter 2.
2. Libkin. Chapter 3.
3. Grädel et al. Section 2.3.

Topics in Logic and Complexity
Part 7

Anuj Dawar

MPhil Advanced Computer Science, Lent 2013

Connectivity

Consider the signature $(E, <)$.

Consider structures $G = (V, E, <)$ in which E is a graph relation and $<$ is a linear order.

There is no first order sentence γ in this signature such that

$$G \models \gamma \text{ if, and only if, } (V, E) \text{ is connected.}$$

Proof

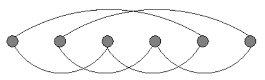
Suppose there was such a formula γ.

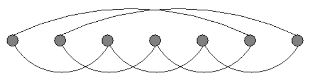
Let γ' be the formula obtained by replacing every occurrence of $E(x, y)$ in γ by the following formula

$$\begin{aligned}&y = x + 2 \vee \\ &(x = \max \wedge y = \min + 1) \vee \\ &(y = \min \wedge x = \max - 1).\end{aligned}$$

Then, $\neg\gamma'$ defines evenness on linear orders!

Proof





We obtain two *disjoint* cycles on linear orders of even length, and a *single* cycle on linear orders of odd length.

Reduction

The above is, in fact, a *first-order definable reduction* from the problem of evenness of linear orders to the problem of connectivity of ordered graphs.

It follows from the above that there is no first order formula that can express the *transitive closure* query on graphs.

Any such formula would also work on ordered *graphs.*

Gaifman Graphs and Neighbourhoods

On a structure $\mathbb{A}$, define the binary relation:

> $E(a_1, a_2)$ if, and only if, there is some relation R and some tuple $\mathbf{a}$ containing both a_1 and a_2 with $R(\mathbf{a})$.

The graph $G\mathbb{A} = (A, E)$ is called the *Gaifman graph* of $\mathbb{A}$.

$dist(a, b)$ — the distance between a and b in the graph (A, E).

$\mathrm{Nbd}_r^{\mathbb{A}}(a)$ — the substructure of $\mathbb{A}$ given by the set:

$$\{b \mid dist(a, b) \leq r\}$$

Hanf Locality Theorem

We say $\mathbb{A}$ and $\mathbb{B}$ are *Hanf equivalent* with radius r ($\mathbb{A} \simeq_r \mathbb{B}$) if, for every $a \in A$ the two sets

$$\{a' \in a \mid \mathrm{Nbd}_r^{\mathbb{A}}(a) \cong \mathrm{Nbd}_r^{\mathbb{A}}(a')\} \quad \text{and} \quad \{b \in B \mid \mathrm{Nbd}_r^{\mathbb{A}}(a) \cong \mathrm{Nbd}_r^{\mathbb{B}}(b)\}$$

have the same cardinality

and, similarly for every $b \in B$.

Theorem (Hanf)
For every vocabulary σ and every p there is $r \leq 3^p$ such that for any σ-structures $\mathbb{A}$ and $\mathbb{B}$: if $\mathbb{A} \simeq_r \mathbb{B}$ then $\mathbb{A} \equiv_p \mathbb{B}$.

In other words, if $r \geq 3^p$, the equivalence relation $\simeq_r$ is a refinement of $\equiv_p$.

Hanf Locality

Duplicator's strategy is to maintain the following condition:

After k moves, if $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ have been selected, then

$$\bigcup_i \mathrm{Nbd}_{3^{p-k}}^{\mathbb{A}}(a_i) \cong \bigcup_i \mathrm{Nbd}_{3^{p-k}}^{\mathbb{B}}(b_i)$$

If *Spoiler* plays on a within distance $2 \cdot 3^{p-k-1}$ of a previously chosen point, play according to the isomorphism, otherwise, find b such that

$$\mathrm{Nbd}_{3^{p-k-1}}(a) \cong \mathrm{Nbd}_{3^{p-k-1}}(b)$$

and b is not within distance $2 \cdot 3^{p-k-1}$ of a previously chosen point.

Such a b is guaranteed by $\simeq_r$.

Uses of Hanf locality

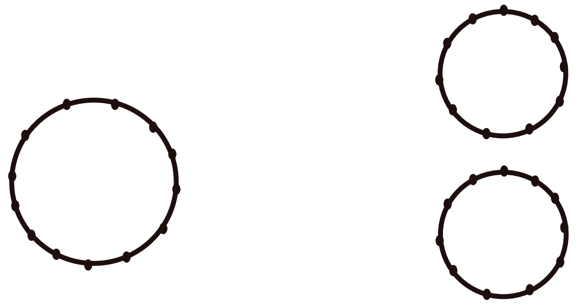
The Hanf locality theorem immediately yields, as special cases, the proofs of undefinability of:

- *connectivity*;
- *2-colourability*
- *acyclicity*
- *planarity*

A simple illustration can suffice.

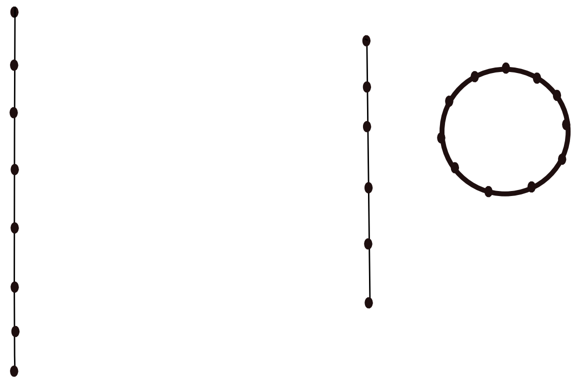
Connectivity

To illustrate the undefinability of *connectivity* and *2-colourability*, consider on the one hand the graph consisting of a single cycle of length $4r + 6$ and, on the other hand, a graph consisting of two disjoint cycles of length $2r + 3$.



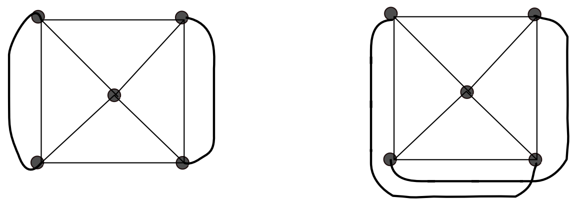
Acyclicity

A figure illustrating that *acyclicity* is not first-order definable.



Planarity

A figure illustrating that *planarity* is not first-order definable.



Monadic Second Order Logic

MSO consists of those second order formulas in which all relational variables are *unary*.

> *That is, we allow quantification over* sets *of elements, but not other relations.*

Any MSO formula can be put in prenex normal form with second-order quantifiers preceding first order ones.

Mon.Σ^1_1 — MSO formulas with only *existential* second-order quantifiers in prenex normal form.

Mon.Π^1_1 — MSO formulas with only *universal* second-order quantifiers in prenex normal form.

Undefinability in MSO

The method of games and *locality* can also be used to show *inexpressibility* results in MSO.

In particular,

There is a Mon.Σ^1_1 query that is not definable in Mon.Π^1_1

(Fagin 1974)

Note: A similar result without the *monadic* restriction would imply that NP $\neq$ co-NP and therefore that P $\neq$ NP.

Connectivity

Recall that *connectivity* of graphs can be defined by a Mon.Π^1_1 sentence.

$$\forall S(\exists x\, Sx \wedge (\forall x \forall y\, (Sx \wedge Exy) \rightarrow Sy)) \rightarrow \forall x\, Sx$$

and by a Σ^1_1 sentence (simply because it is in NP).

We now aim to show that *connectivity* is not definable by a Mon.Σ^1_1 sentence.

MSO Game

The m-round monadic Ehrenfeucht game on structures $\mathbb{A}$ and $\mathbb{B}$ proceeds as follows:

- At the ith round, *Spoiler* chooses one of the structures (say $\mathbb{B}$) and plays either a point move or a set move.

  In a point move, he chooses one of the elements of the chosen structure (say b_i) – *Duplicator* must respond with an element of the other structure (say a_i).
  In a set move, he chooses a subset of the universe of the chosen structure (say S_i) – *Duplicator* must respond with a subset of the other structure (say R_i).

MSO Game

- If, after m rounds, the map

  $$a_i \mapsto b_i$$

  is a partial isomorphism between

  $$(\mathbb{A}, R_1, \ldots, R_q) \text{ and } (\mathbb{B}, S_1, \ldots, S_q)$$

  then *Duplicator* has won the game, otherwise *Spoiler* has won.

MSO Game

If we define the *quantifier rank* of an MSO formula by adding the following inductive rule to those for a formula of FO:

if $\phi = \exists S \psi$ or $\phi = \forall S \psi$ then $\mathrm{qr}(\phi) = \mathrm{qr}(\psi) + 1$

then, we have

Duplicator has a winning strategy in the m-round monadic Ehrenfeucht game on structures $\mathbb{A}$ and $\mathbb{B}$ if, and only if, for every sentence ϕ of MSO with $\mathrm{qr}(\phi) \leq m$

$$\mathbb{A} \models \phi \quad \text{if, and only if,} \quad \mathbb{B} \models \phi$$

Existential Game

The m, p-move existential game on $(\mathbb{A}, \mathbb{B})$:

- First *Spoiler* makes m set moves on $\mathbb{A}$, and *Duplicator* replies on $\mathbb{B}$.
- This is followed by an Ehrenfeucht game with p point moves.

If *Duplicator* has a winning strategy, then for every Mon.Σ^1_1 sentence:

$$\phi \equiv \exists R_1 \ldots \exists R_m \, \psi$$

with $\mathrm{qr}(\psi) = p$,

$$\text{if } \mathbb{A} \models \phi \text{ then } \mathbb{B} \models \phi$$

Variation

To show that a Boolean query Q is not Mon.Σ^1_1 definable, find for each m and p

- $\mathbb{A} \in Q$; and
- $\mathbb{B} \not\in P$; such that
- *Duplicator* wins the m, p move game on $(\mathbb{A}, \mathbb{B})$.

Or,

- *Duplicator* chooses $\mathbb{A}$.
- *Spoiler* colours $\mathbb{A}$ (with 2^m colours).
- *Duplicator* chooses $\mathbb{B}$ and colours it.
- They play a p-round Ehrenfeucht game.

Application

Write C_n for the graph that is a simple *cycle* of length n.

For n sufficiently large, and any *colouring* of C_n, we can find an $n' < n$ and a colouring of

$C_{n'} \oplus C_{n-n'}$ the disjoint union of two cycles—one of length n', the other of length $n - n'$

So that the graphs C_n and $C_{n'} \oplus C_{n-n'}$ are $\simeq_r$ equivalent.

Taking $n > (2r+1)^{2^m+2}$ suffices.

Reading List for this Part

1. Ebbinghaus and Flum. Section 2.4.
2. Libkin. Chapter 4.
3. Grädel et al. Section 2.3 and 2.5

Topics in Logic and Complexity Part 8

Anuj Dawar

MPhil Advanced Computer Science, Lent 2013

Expressive Power of Logics

We have seen that the expressive power of *first-order logic*, in terms of computational complexity is *weak*.

Second-order logic allows us to express all properties in the *polynomial hierarchy*.

Are there interesting logics intermediate between these two?

We have seen one—*monadic second-order logic*.

We now examine another—*LFP*—the logic of *least fixed points*.

Inductive Definitions

LFP is a logic that formalises *inductive definitions*.

> Unlike in second-order logic, we cannot quantify over *arbitrary* relations, but we can build new relations *inductively*.

Inductive definitions are pervasive in mathematics and computer science.

The *syntax* and *semantics* of various formal languages are typically defined inductively.

> *viz.* the definitions of the syntax and semantics of first-order logic seen earlier.

Transitive Closure

The *transitive closure* of a binary relation E is the *smallest* relation T satisfying:

- $E \subseteq T$; and
- if $(x, y) \in T$ and $(y, z) \in E$ then $(x, z) \in T$.

This constitutes an *inductive definition* of T and, as we have already seen, there is no *first-order* formula that can define T in terms of E.

Monotone Operators

In order to introduce LFP, we briefly look at the theory of *monotone operators*, in our restricted context.

We write $\mathsf{Pow}(A)$ for the powerset of A.

An operator in A is a function

$$F : \mathsf{Pow}(A) \to \mathsf{Pow}(A).$$

F is *monotone* if

$$\text{if } S \subseteq T, \text{ then } F(S) \subseteq F(T).$$

Least and Greatest Fixed Points

A *fixed point* of F is any set $S \subseteq A$ such that $F(S) = S$.

S is the *least fixed point* of F, if for all fixed points T of F, $S \subseteq T$.

S is the *greatest fixed point* of F, if for all fixed points T of F, $T \subseteq S$.

Least and Greatest Fixed Points

For any monotone operator F, define the collection of its *pre-fixed points* as:

$$Pre = \{S \subseteq A \mid F(S) \subseteq S\}.$$

Note: $A \in Pre$.

Taking

$$L = \bigcap Pre,$$

we can show that L is a fixed point of F.

Fixed Points

For any set $S \in \mathit{Pre}$,

$L \subseteq S$	by definition of L.
$F(L) \subseteq F(S)$	by monotonicity of F.
$F(L) \subseteq S$	by definition of Pre.
$F(L) \subseteq L$	by definition of L.
$F(F(L)) \subseteq F(L)$	by monotonicity of F
$F(L) \in \mathit{Pre}$	by definition of Pre.
$L \subseteq F(L)$	by definition of L.

Least and Greatest Fixed Points

L is a *fixed point* of F.

Every fixed point P of F is in Pre, and therefore $L \subseteq P$.

Thus, L is the least fixed point of F

Similarly, the greatest fixed point is given by:

$$G = \bigcup \{S \subseteq A \mid S \subseteq F(S)\}.$$

Iteration

Let A be a *finite* set and F be a *monotone* operator on A.

Define for $i \in \mathbb{N}$:

$$\begin{aligned} F^0 &= \emptyset \\ F^{i+1} &= F(F^i). \end{aligned}$$

For each i, $F^i \subseteq F^{i+1}$ (proved by induction).

Iteration

Proof by induction.

$$\emptyset = F^0 \subseteq F^1.$$

If $F^i \subseteq F^{i+1}$ then, by monotonicity

$$F(F^i) \subseteq F(F^{i+1})$$

and so $F^{i+1} \subseteq F^{i+2}$.

Fixed-Point by Iteration

If A has n elements, then

$$F^n = F^{n+1} = F^m \quad \text{for all} \quad m > n$$

Thus, F^n is a fixed point of F.

Let P be any fixed point of F. We can show induction on i, that $F^i \subseteq P$.

$$F^0 = \emptyset \subseteq P$$

If $F^i \subseteq P$ then

$$F^{i+1} = F(F^i) \subseteq F(P) = P.$$

Thus F^n is the *least fixed point* of F.

Defined Operators

Suppose ϕ contains a relation symbol R (of arity k) not interpreted in the structure $\mathbb{A}$ and let $\mathbf{x}$ be a tuple of k free variables of ϕ.

For any relation $P \subseteq A^k$, ϕ defines a new relation:

$$F_P = \{\mathbf{a} \mid (\mathbb{A}, P) \models \phi[\mathbf{a}]\}.$$

The operator $F_\phi : \mathsf{Pow}(A^k) \to \mathsf{Pow}(A^k)$ defined by ϕ is given by the map

$$P \mapsto F_P.$$

Or, $F_{\phi,\mathbf{b}}$ if we fix parameters $\mathbf{b}$.

Positive Formulas

Definition
A formula ϕ is *positive* in the relation symbol R, if every occurence of R in ϕ is within the scope of an even number of negation signs.

Lemma
For any structure $\mathbb{A}$ not interpreting the symbol R, any formula ϕ which is positive in R, and any tuple $\mathbf{b}$ of elements of A, the operator $F_{\phi,\mathbf{b}} : \mathsf{Pow}(A^k) \to \mathsf{Pow}(A^k)$ is monotone.

Reading List for this Part

1. Ebbinghaus and Flum. Section 8.1.
2. Libkin. Sections 10.1 and 10.2.
3. Grädel et al. Section 3.3.